```
{ -- FLORIDA HIGH SCHOOLS COMPUTING COMPETITION '94 }
{ -- PASCAL PROGRAM SOLUTIONS }


{1.1}
program One1T94;
{ -- This program will display the 1994 FHSCC sponsors. }
  var
    I: Integer;

begin
  Writeln ('FHSCC ''94 IS SPONSORED BY:');
  Writeln;
  for I := 1 to 4 do
    Writeln ('GTEDS  GTEDS  GTEDS  GTEDS  GTEDS');
  Writeln;
  for I := 1 to 4 do
    Writeln ('USF CENTER FOR EXCELLENCE');
  Writeln;
  for I := 1 to 4 do
    Writeln ('FLORIDA DEPARTMENT OF EDUCATION');
end.



{1.2}
program One2T94;
{ -- This program will determine if an applicant is hired. }
  var
    Entrance, Offer: String[8];

begin
  Write ('Entrance requirement: ');  Readln (Entrance);
  Write ('Plans to accept or reject offer: ');  Readln (Offer);
  Write ('APPLICANT WILL ');
  if (Entrance <> 'PASSED') or (Offer <> 'ACCEPT') then
    Write ('NOT ');
  Writeln ('BE HIRED');
end.


{1.3}
program One3T94;
{ -- This program will display number of employees. }
  var
    Current, Hiring, Leaving, Total: LongInt;

begin
  Write ('Enter current number: ');  Readln (Current);
  Write ('Enter number hiring: ');   Readln (Hiring);
  Write ('Enter number leaving: ');  Readln (Leaving);
  Total := Current + Hiring - Leaving;
  Writeln (Total, ' EMPLOYEES');
end.
```

```
{1.4}
program One4T94;
{ -- This program will total the millions converted. }
  var
    Num, Sum: Real;
    Million:  String[10];

begin
  Sum := 0;
  Write ('Enter number of accounts: ');  Readln (Num, Million);
  While Num > -999 do begin
    Sum := Sum + Num;
    Write ('Enter number of accounts: '); Readln (Num, Million);
  end;
  Sum := Sum + 0.00001;  { -- error factor of computer }
  if Sum - int(Sum) < 0.001 then
    Write (Trunc(Sum))
  else
    Write (Sum: 3:1);
  Writeln (' MILLION ACCOUNTS CONVERTED TO CBSS');
end.


{1.5}
program One5T94;
{ -- This program will compute the gross wages earned. }
  var
    Hours, Rate, OverTime, Wages: Real;

begin
  Write ('Enter hours, rate: ');  Readln (Hours, Rate);
  if Hours > 40 then
    Hours := Hours + (Hours - 40) * 0.5;
  Wages := Hours * Rate;
  Writeln ('GROSS WAGES ARE $', Wages :5:2);
end.
```

```
{1.6}
program One6T94;
{ -- This program will tally the number of customers sold. }
  var
    AreaCode, Num, I: Integer;
    Sum:              LongInt;

begin
  Write ('Enter number of area codes: ');  Readln (Num);
  Sum := 0;
  for I := 1 to Num do begin
    Write ('Enter area code: ');  Readln (AreaCode);
    Case AreaCode of
      706:  Inc(Sum, 95000);
      208:  Inc(Sum, 54321);
      912:  Inc(Sum, 99825);
      605:  Inc(Sum, 88776);
      404:  Inc(Sum, 90175);
    end;
  end;
  Writeln ('TOTAL NUMBER OF ACCOUNTS BEING SOLD = ', Sum);
end.

{1.7}
program One7T94;
{ -- This program will display the cost to fix error in phase. }
  const
    Phases: Array[1..6] of String[15] =  ('REQUIREMENTS',
       'DESIGN', 'CODING', 'SYSTEM TEST', 'ACCEPTANCE TEST',
       'MAINTENANCE');
    Factor: Array [1..6] of Integer = (1, 5, 10, 20, 50, 100);
  var
    I, Cost: Integer;
    Phase:   String[15];

begin
  Write ('Enter cost $: ');  Readln (Cost);
  Write ('Enter phase: ');   Readln (Phase);
  I := 1;
  while Phase <> Phases[I] do Inc(I);
  Write ('COST IS $', Cost * Factor[I]);
  Writeln (' TO FIX PROBLEM IN ', Phase, ' PHASE');
end.

{1.8}
program One8T94;
{ -- This program will compute the maximum blocksize. }
  var
    LRecL, Num: Integer;
begin
  Write ('Enter logical record length: ');  Readln (LRecL);
  Num := 23476 div LRecL;
  Writeln ('BLOCKSIZE = ', LRecL * Num, ' BYTES');
end.
```

```
{1.9}
program One9T94;
{ -- This program will compute an electric bill. }
  var
    Hours, Bill, Rate: Real;

begin
  Write ('Enter kilowatt hours: ');  Readln (Hours);
  if Hours < 10.0 then
    Rate := 4.95
  else
    Rate := 5.65;
  Bill := Rate * Hours;
  Bill := Bill * (1 + 0.03 + 0.06);
  if Hours > 30.0 then
    Bill := Bill + 25.0;
  Writeln ('THE CUSTOMER''S BILL IS $', Bill :3:2);
end.


{1.10}
program One10T94;
{ -- This program will determin if a 5x5 matrix is symmetric. }
  var
    A:         Array[1..5, 1..5] of Integer;
    I, J:      Integer;
    Symmetric: Boolean;

begin
  for I := 1 to 5 do begin
    Write ('Enter row: ');
    Readln (A[I,1], A[I,2], A[I,3], A[I,4], A[I,5]);
  end;
  Symmetric := True;
  for I := 1 to 5 do
    for J := 1 to 5 do
      if A[I, J] <> A[J, I] then
        Symmetric := False;

  Write ('MATRIX IS ');
  if not Symmetric then
    Write ('NOT ');
  Writeln ('SYMMETRIC');
end.
```

```
{2.1}
program Two1T94;
{ -- This program will simulate NTF's ESP utility. }
  var
    Jobs:          String[50];
    Job:           Array[1..20] of String[2];
    I, L, LastCK: Integer;
    OK:            String[1];

begin
  Write ('Enter jobs/CK: ');  Readln (Jobs);
  L := (Length(Jobs) + 1) div 3;
  for I := 1 to L do
    Job[I] := Copy(Jobs, I*3 - 2, 2);

  LastCK := 0;
  repeat
    I := LastCK + 1;
    while Job[I] <> 'CK' do begin
      Writeln (Job[I]);
      Inc(I);
    end;
    Writeln ('EVERYTHING OK?' );  Readln (OK);
    if OK = 'N' then
      I := LastCK
    else
      LastCK := I;
  until I = L;
end.
```

```
{2.2}
program Two2T94;
{ -- This program will display random letters in random areas. }
  uses Crt;
  var
    Letter, LastLet, Ch: Char;
    R, C:                Integer;

begin
  Randomize;
  Ch := ' ';
  repeat
    ClrScr;
    if Ch = ' ' then begin
        Letter := Chr(65 + Random(26));
        Ch := Letter;
      end
    else
      Letter := Ch;
    LastLet := Letter;
    repeat
      R := Random(23) + 1;  C := Random(79) + 1;
      GotoXY (C, R);  Write (Letter);
      Delay(100);
      if Keypressed then
        Ch := UpCase(ReadKey);
    until (Ch <> LastLet);
  until (Ch <> ' ') and ((Ch < 'A') or (Ch > 'Z'));
end.
```

```
{2.3}
program Two3T94;
{ -- This program will transliterate Hebrew to English. }
  var
    St, Trans:  String[80];
    I:          Integer;
    Ch, LastCh: String[1];
    Let:        String[2];

begin
  Write ('Enter letters: ');  Readln (St);
  LastCh := ' ';  Trans := '';
  for I := 1 to Length(St) do begin
    Ch := Copy(St, I, 1);  Let := Ch;
    if LastCh = ' ' then begin
      if Ch = 'A' then
        if Copy(St, I+1, 1) = 'L' then
          Let := ')'
        else
          Let := '(';
      if Copy(St, I, 3) = 'HET' then Let := 'CH';
      if Copy(St, I, 2) = 'TS'  then Let := 'TS';
      Trans := Let + Trans;
    end;
    LastCh := Ch;
  end;
  Writeln (Trans);
end.
```

```
{2.4}
program Two4T94;
{ -- This program will append a "security digit" to an account }
  var
    Acct:  String[15];
    Ch:    String[1];
    Error: Boolean;
    Sum, I, L, Dig, Code: Integer;

begin
  Write ('Enter account number: ');  Readln (Acct);
  L := Length (Acct);
  Error := False;
  if (L <> 7) and (L <> 9) then begin
    Writeln ('ERROR - INCORRECT LENGTH');
    Error := True;
  end;

  { -- Sum the valid digits }
  Sum := 0;
  for I := 1 to L do begin
    Ch := Copy(Acct, I, 1);
    Val (Ch, Dig, Code);
    if (Dig = 0) and (Ch <> '0') then begin
      Writeln ('ERROR - NON-NUMERIC');
      Exit;
    end;
    Sum := Sum + Dig;
  end;

  { -- If account is valid, append security digit }
  if not Error then begin
    Write (Acct);
    if Sum mod 2 = 0 then
      Writeln ('1')
    else
      Writeln ('0');
  end;
end.
```

```
{2.5}
program Two5T94;
{ -- This program will count the digits used in a book. }
  var
    I, J, LPage, M, Dig, Code, Max, Min: Integer;
    A:    Array[0..9] of Integer;
    Page: String[4];

begin
  Write ('Enter last page: ');  Readln (LPage);
  Write ('Enter M: ');          Readln (M);
  for I := 0 to 9 do A[I] := 0;

  for I := 2 to LPage do begin
    if (I mod M > 0) then begin
      Str (I, Page);
      for J := 1 to Length(Page) do begin
        Val (Copy(Page, J, 1), Dig, Code);
        Inc(A[Dig]);
      end;
    end;
  end;
  Max := 0;  Min := 32000;
  for I := 0 to 9 do begin
    Writeln (I, ' APPEARS ', A[I], ' TIMES');
    if A[I] > Max then Max := A[I];
    if A[I] < Min then Min := A[I];
  end;

  Writeln;
  Write ('DIGIT(S) APPEARING THE MOST: ');
  for I := 0 to 9 do
    if A[I] = Max then Write (I, ' ');
  Writeln;
  Write ('DIGIT(S) APPEARING THE LEAST: ');
  for I := 0 to 9 do
    if A[I] = Min then Write (I, ' ');
end.
```

```
{2.6}
program Two6T94;
{ -- This program will compute the roots for a quadratic. }
  var
    A, B, C, D, R1, R2: Integer;

begin
  Write ('Enter coefficients A, B, C: ');  Readln (A, B, C);
  D  := B * B - 4 * A * C;
  Write ('THE ROOTS ARE ');
  if D >= 0 then
    begin
      Writeln ('REAL');
      R1 := (-B + Trunc(Sqrt(D))) div (2 * A);
      R2 := (-B - Trunc(Sqrt(D))) div (2 * A);
      if D > 0 then
        Writeln ('THE ROOTS ARE ', R1, ' AND ', R2)
      else
        Writeln ('THE ONLY ROOT IS ', R1);
    end
  else  { -- D < 0    Roots are Complex }
    begin
      Writeln ('COMPLEX');
      R1 := -B div (2 * A);
      R2 := Trunc(Sqrt(-D)) div (2 * A);
      Write ('THE ROOTS ARE ', R1, ' + ', R2, 'I AND ');
      Writeln (R1, ' - ', R2, 'I');
    end;
end.
```

```
{2.7}
program Two7T94;
{ -- This program will generate 5 customer account numbers. }
  const
    Num: Integer = 15;
  var
    Seed: Real;
    I, J, Dig, Code, Sum, CheckDig: Integer;
    Cust: String[10];
    Temp: String[1];

begin
  Write ('Enter seed used last: ');  Readln (Seed);
  I := 0;
  while I < Num do begin
    { -- Add 1 and reverse last 2 digits }
    Seed := Seed + 1;
    Str (Seed :9:0, Cust);

    Temp := Cust[9];  Cust := Copy(Cust, 1, 8);
    Insert (Temp, Cust, 8);

    for J := 1 to 9 do
      if Cust[J] = ' ' then Cust[J] := '0';
    { -- Shift digits 3-9 and insert last 2 digits }
    Cust := Copy(Cust, 1, 2) + Copy(Cust, 8, 2) +
            Copy(Cust, 3, 5);
    { -- Calculate Check Digit }
    Sum := 0;
    for J := 1 to 9 do begin
      Val (Copy(Cust, J, 1), Dig, Code);
      Sum := Sum + Dig * (11 - J);
    end;
    CheckDig := 11 - (Sum mod 11);
    if CheckDig = 11 then CheckDig := 0;
    if CheckDig <> 10 then begin
      Writeln (Cust, CheckDig);
      Inc(I);
    end;
  end;  { -- while }
end.
```

```
{2.8}
program Two8T94;
{ -- This program will compute speed, distance, time. }
  var
    S, D, T, HH, MM: Real;
    Tim:             String[6];
    Ttype:           String[1];
    L, Code:         Integer;

begin
  Write ('Enter speed, distance: ');  Readln (S, D);
  Write ('Enter time: ');  Readln (Tim);
  if Tim <> '0' then begin
    L := Length(Tim);
    Ttype := Copy(Tim, L, 1);
    if (TType = 'H') or (TType = 'M') then
      Val(Copy(Tim, 1, L-1), T, Code)
    else { -- Ttype = 'C' }
      begin
        Val (Copy(Tim,1,2), HH, Code);
        Val (Copy(Tim,4,2), MM, Code);
        T := HH + MM / 60;
      end;
    if Ttype = 'M' then
      T := T / 60;
  end;

  if S = 0 then
      Writeln ('SPEED = ', D / T :5:1, ' MPH')
  else if D = 0 then
      Writeln ('DISTANCE = ', S * T :6:1, ' MILES')
  else if Tim = '0' then
      Writeln ('TIME = ', D / S :4:2, ' HOURS');
end.
```

```
{2.9}
program Two9T94;
{ -- This program will compute the response time. }
  var
    RDate, CDate, RTime, CTime:           String[8];
    RDay, CDay, RMin, RHour, CMin, CHour: Byte;
    Code, Res:                            Integer;

begin
  Write ('Enter reported date: ');  Readln (RDate);
  Write ('Enter reported time: ');  Readln (RTime);
  Write ('Enter cleared date: ');   Readln (CDate);
  Write ('Enter cleared time: ');   Readln (CTime);

  Val(Copy(RDate, 4, 2), RDay, Code);
  Val(Copy(CDate, 4, 2), CDay, Code);
  Val(Copy(RTime, 1, 2), RHour, Code);
  Val(Copy(RTime, 4, 2), RMin, Code);
  Val(Copy(CTime, 1, 2), CHour, Code);
  Val(Copy(CTime, 4, 2), CMin, Code);

  Res := 0;
  if RHour < 8 then begin
    RHour := 8;  RMin := 0;
  end;
  if CHour < 8 then begin
    CHour := 8;  CMin := 0;
  end;
  if CHour >= 17 then begin
    CHour := 17;  CMin := 0;
  end;
  if RHour >=17 then begin
    RHour := 17;  RMin := 0;
  end;

  Res := (CDay - RDay) * 9 * 60;
  Res := Res + (CHour - RHour) * 60 + (CMin - RMin);

  Writeln ('RESPONSE TIME WAS ', Res, ' MINUTES');
end.
```

```
{2.10}
program Two10T94;
{ -- This program will display the discounts for calling plans }
  var
    OrigNum, ToNum:                    String[10];
    Handicap, OrigArea, ToArea:        String[3];
    CallLen, Cost, PlanA, PlanB, PlanC: Real;

begin
  Write ('Enter originating number: ');  Readln (OrigNum);
  Write ('Enter number called: ');  Readln (ToNum);
  Write ('Handicapped person?: ');  Readln (Handicap);
  Write ('Enter length of call: '); Readln (CallLen);
  Write ('Enter cost of call $: '); Readln (Cost);

  PlanA := 9E9;  PlanB := 9E9;  PlanC := 9E9;
  OrigArea := Copy(OrigNum, 1, 3);
  ToArea   := Copy(ToNum, 1, 3);
  if (CallLen >= 5.0) and (OrigArea <> ToArea) then begin
    PlanA := Cost * 0.85;
    Writeln ('THE PLAN A CHARGE WOULD BE $', PlanA :3:2);
  end;
  if Handicap = 'YES' then begin
    PlanB := Cost * 0.90;
    Writeln ('THE PLAN B CHARGE WOULD BE $', PlanB :3:2);
  end;
  if (ToArea = '407') and (OrigArea <> ToArea)
  and (CallLen >= 3.5) then begin
    PlanC := Cost * 0.8775;
    Writeln ('THE PLAN C CHARGE WOULD BE $', PlanC :3:2);
  end;

  if (PlanA = 9E9) and (PlanB = 9E9) and (PlanC = 9E9) then
    Writeln ('THIS PERSON DOES NOT QUALIFY FOR ANY PLANS')
  else begin
    Write ('THIS PERSON WOULD RECEIVE PLAN ');
    if (PlanA < PlanB) and (PlanA < PlanC) then
      Writeln ('A')
    else
      if (PlanB < PlanA) and (PlanB < PlanC) then
         Writeln ('B')
       else
         Writeln ('C');
  end;
end.
```

```
{3.1}
program Thr1T94;
{ -- This program will convert transliterated English to Greek. }
{ -- The Greek letters ETA and OMICRON are not used. }
{ -- The Greek letter THETA is placed at the end of the list. }
  const
    Name: Array [1..24] of String[8] = ('ALPHA', 'BETA', 'GAMMA',
      'DELTA', 'EPSILON', 'ZETA', '-TA',  'IOTA',
      'KAPPA', 'LAMBDA', 'MU', 'NU', 'XI', '-MICRON', 'PI',
      'RHO', 'SIGMA', 'TAU', 'UPSILON', 'PHI', 'CHI', 'PSI',
      'OMEGA', 'THETA');
    Value: Array [1..24] of Integer = (1, 2, 3, 4, 5, 7, 8,
      10, 20, 30, 40, 50, 60, 70, 80,
      100, 200, 300, 400, 500, 600, 700, 800, 9);
  var
    I, J, Sum, Inc: Integer;
    Trans:          String[15];
    Ch:             String[2];

begin
  Write ('Enter transliteration: ');  Readln (Trans);
  Sum := 0;
  I := 1;
  while I <= Length(Trans) do begin
    Ch := Copy(Trans, I, 2);
    if (Ch = 'TH') or (Ch = 'PH') or (Ch = 'CH') or (Ch = 'PS')
    then
      Inc := 2
    else
      Inc := 1;
    J := 1;
    while Copy(Trans, I, Inc) <> Copy(Name[J], 1, Inc) do
      J := J + 1;
    Write (Name[J], ' ');
    Sum := Sum + Value[J];
    I := I + Inc;
  end;  { -- While I }
  Writeln;  Writeln ('NUMERICAL SUM = ', Sum);
end.


{3.2}
program Thr2T94;
{ -- This program will move a taxi in a grid. }
  const
    South: Integer = 8;
  var
    Num, SNum, NumLet, SNumLet: Integer;
    SLet, Dir:   Char;
    Out, TooFar: Boolean;

begin
  Write ('Enter starting position: ');  Readln (SLet, SNum);
  Num := SNum;
  SNumLet := Ord(SLet) - Ord('A') + 1;  NumLet := SNumLet;
```

```
repeat
  Write ('Enter direction: ');  Readln (Dir);
  Out := False;  TooFar := False;
  Case Dir of
    'N':  if Num = 1 then
            Out := True
          else
            if SNum - 2 = Num then
              TooFar := True
            else
              Dec(Num);
    'S':  if Num = South then
            Out := True
          else
            if SNum + 2 = Num then
              TooFar := True
            else
              Inc(Num);
     'W':  if NumLet = 1 then
             Out := True
           else
             if SNumLet - 2 = NumLet then
               TooFar := True
             else
               Dec(NumLet);
     'E':  if NumLet = 26 then
             Out := True
           else
             if SNumLet + 2 = NumLet then
               TooFar := True
             else
               Inc(NumLet);
    end;  { -- case }

    if Out then
      Writeln ('LOCATION IS OUTSIDE CITY LIMITS')
    else if TooFar then
      begin
        Write ('LOCATION IS TOO FAR ');
        Case Dir of
          'N':  Writeln ('NORTH');
          'S':  Writeln ('SOUTH');
          'W':  Writeln ('WEST');
          'E':  Writeln ('EAST');
        end;
      end
    else
      if Dir <> 'Q' then begin
        Write ('TAXI LOCATION IS ');
        Writeln (Chr(NumLet + 64), ',', Num);
      end;
   until Dir = 'Q';
 end.
```

```
{3.3}
program Thr3T94;
{ -- This program will display anagrams. }
  var
    W, W2: Array [1..9] of String[7];
    SortW: Array [1..7] of String[1];
    I, J, K, L, Num, Tot:  Integer;
    T:                     String[7];

begin
  Write ('Enter number of words: ');  Readln (Num);
  for I := 1 to Num do begin
    Write ('Enter word: ');  Readln (W[I]);
  end;

  { -- Sort words in ascending order }
  for I := 1 to Num - 1 do
    for J := I + 1 to Num do
      if W[I] > W[J] then begin
        T := W[I];  W[I] := W[J];  W[J] := T;
      end;

  { -- Sort letters within word and store in W2[] }
  for I := 1 to Num do begin
    L := Length(W[I]);
    for J := 1 to L do
      SortW[J] := Copy(W[I], J, 1);
    for J := 1 to L - 1 do
      for K := J + 1 to L do
        if SortW[J] > SortW[K] then begin
          T := SortW[J];  SortW[J] := SortW[K];
          SortW[K] := T;
        end;

    W2[I] := '';
    for J := 1 to L do
      W2[I] := W2[I] + SortW[J];
  end;

  { -- Compare every pair of sorted words for a match. }
  Tot := 0;
  for I := 1 to Num - 1 do
    for J := I + 1 to Num do
      if W2[I] = W2[J] then begin
        Tot := Tot + 1;
        if Tot = 1 then
          Write ('ANAGRAMS: ')
        else
          Write ('          ');
        Writeln (W[I], ', ', W[J])
      end;
  if Tot = 0 then
    Writeln ('NO ANAGRAMS IN LIST');
end.
```

```
{3.4}
program Thr4T94;
{ -- This program will place money in envelopes. }
  var
    Money, A, B, C, D, Incr, Total: Integer;

begin
  Write ('Enter amount of money: '); Readln (Money);
  Total := 0;
  Incr := Money div 2;
  for A := 1 to Incr - 2 do
    for B := A + 1 to Incr - 1 do
      for C := B + 1 to Incr do begin
        { -- D will contain the largest amount to disperse }
        D := Money - A - B - C;
        if (A < B) and (B < C) and (C < D) then begin
          Write ('TAKE ', A, ' ', B, ' ', C, ' ', D);
          { -- (D - A) dollars are dispersed to make }
          { --         A=B, B=C, C=D, and D=A        }
          Write (' AND DISPERSE ', D - A, ' DOLLARS TO MAKE ');
          Writeln (B, ' ', C, ' ', D, ' ', A);
          Inc(Total);
        end;
      end;
   Writeln ('TOTAL NUMBER OF SOLUTIONS = ', Total);
end.
```

```
{3.5}
program Thr5T94;
{ -- This program will convert Gregorian and Julian dates. }
  const
    Month: Array [1..12] of Byte =
      (31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31);
  var
    MDays: Array [1..12] of Byte;
    M, D, Y, I, Days, Code: Integer;
    DType, Date, YY:        String[11];

begin
  for I := 1 to 12 do MDays[I] := Month[I];
  Write ('Enter Julian or Gregorian: ');  Readln (DType);
  Write ('Enter date: ');  Readln (Date);
  if DType = 'GREGORIAN' then
    { -- Convert Gregorian to Julian }
    begin
      Val(Copy(Date, 1, 2), M, Code);
      Val(Copy(Date, 4, 2), D, Code);
      YY := Copy(Date, 7, 2);
      Val(YY, Y, Code);
      Days := D;
      for I := 1 to M - 1 do
        Days := Days + MDays[I];
      if (Y mod 4 = 0) and (M > 2) then
        Inc(Days);
      Write ('JULIAN DATE = ', YY);
      if Days < 100 then Write ('0');
      if Days < 10 then Write ('0');
      Writeln (Days)
    end
  else  { -- Convert Julian to Gregorian }
    begin
      YY := Copy(Date, 1, 2);
      Val(YY, Y, Code);
      Val(Copy(Date, 3, 3), D, Code);
      M := 1;
      if Y mod 4 = 0 then
        MDays[2] := 29;
      while D > MDays[M] do begin
        D := D - MDays[M];
        Inc(M);
      end;
      Write ('GREGORIAN DATE = ');
      if M < 10 then Write ('0');
      Write (M, '/');
      if D < 10 then Write ('0');
      Write (D, '/');
      Writeln (YY);
    end;
end.
```

```
{3.6}
program Thr6T94;
{ -- This program to convert a number for one base to another. }
  var
    Base1, Base2, Num1V, Num2, Power: Real;
    I, J, K, X, Digit:                Byte;
    Num1, NumOut:                     String[9];

begin
  Write ('Enter base of first number: ');  Readln (Base1);
  Write ('Enter number: ');                Readln (Num1);
  Write ('Enter base of output: ');        Readln (Base2);

  { -- Convert Num1 to base 10 number Num1V }
  Num1V := 0;
  for I := 1 to Length(Num1) do begin
    Digit := Ord(Num1[I]) - Ord('0');
    if Digit > 9 then    { -- Digit is a letter digit }
      Dec(Digit, 7);
    Power := 1;
    for J := 1 to Length(Num1) -I do
      Power := Power * Base1;
    Num1V := Num1V + Digit * Power;
  end;

  { -- Convert Num1V to Base2 number }
  NumOut := '';
  J := Trunc(Ln(Num1V) / Ln(Base2));
  for I := J downto 0 do begin
    Power := 1;
    for K := 1 to I do  Power := Power * Base2;
    X := Trunc(Num1V / Power);
      NumOut := Copy ('0123456789ABCDEF', X + 1, 1) + NumOut;
    Num1V := Num1V - X * Power;
  end;
  Writeln (NumOut);
end.
```

```
{3.7}
program Thr7T94;
{ -- This program will SHELL sort numbers generated. }
  const
    Num: Integer = 8000;
    Max: Integer = 7;
  var
    I, J, P, Last, Increment: Integer;
    X:                        Array [-1093..8000] of Real;
    Incr:                     Array [1..7] of Integer;
    Pow, Q, Temp, T:          Real;

begin
  Write ('Enter seed X[0]: ');  Readln (X[0]);
  Pow := 1;
  for I := 1 to 20 do Pow := Pow * 2;
  for I := 1 to Num do begin
    Q := Int ((69069.0 * X[I-1]) / Pow);
    X[I] := 69069.0 * X[I-1] - Pow * Q;
  end;
  for I := -1093 to -1 do X[I] := 0;

  { -- SHELL SORT ROUTINE }
  Incr[Max] := 1;
  { -- Compute Increments }
  for I := Max - 1 downto 1 do Incr[I] := 3 * Incr[I+1] + 1;
  for I := 1 to Max do begin
    Increment := Incr[I];
    for J := 1 to Increment do begin
      Last := Increment + J;
      while Last <= Num do begin
        P := Last;
        T := X[P];
        X[1 - Increment] := T;
        while T < X[P - Increment] do begin
          X[P] := X[P - Increment];
          Dec(P, Increment);
        end;
        X[P] := T;
        Inc(Last, Increment);
      end;
    end;  { -- for J }
  end;  { -- for I }

  { -- Display every 1000th number in ascending order }
  for I := 1 to Num div 1000 do
    Writeln (I*1000, 'TH NUMBER = ', X[I*1000]: 6:0);
end.
```

```
{Alternate solution to 3.7}
program Thr7T94;
{ -- This program will QUICK sort numbers generated. }
  const
    Num: Integer = 8000;
  var
    I:      Integer;
    X:      Array [0..8000] of Real;
    Pow, Q: Real;

procedure Quicksort(L, R: Integer);
{ -- sorts global array X[L..R] where X[R + 1] > any X[L..R] }
  var
    I, J:   Integer;
    T, Piv: Real;

begin
  if L < R then begin
    I := L + 1;  J := R;  Piv := X[L];
    repeat  { -- move pointers I, J inwards as far as possible }
      while X[I] <= Piv do I := I + 1;
      while X[J] > Piv  do J := J - 1;
      if I {still} < J then begin
        { -- Exchange items pointed to by I and J }
        T := X[I];  X[I] := X[J];  X[J] := T;
      end;
    until I > J;
    { -- Now two final replacements finish a partition }
    X[L] := X[J];  X[J] := Piv;
    { -- Finish by recursively sorting left, right partitions }
    Quicksort (L, J-1);  Quicksort (I, R);
  end;  { -- if }
end;  { procedure Quicksort }

begin
  Write ('Enter seed X[0]: ');  Readln (X[0]);
  Pow := 1;
  for I := 1 to 20 do Pow := Pow * 2;
  for I := 1 to Num do begin
    Q := Int ((69069.0 * X[I-1]) / Pow);
    X[I] := 69069.0 * X[I-1] - Pow * Q;
  end;

  Quicksort (1, Num);

  { -- Display every 1000th number in ascending order }
  for I := 1 to Num div 1000 do
    Writeln (I*1000, 'TH NUMBER = ', X[I*1000]: 6:0);
end.
```

```
{3.8}
program Thr8T94;
{ -- This program will compute the volume of a sphere using PI }
  const
    PI1: String[37] = '31415926535897932384626433832795028884';
    PI2: String[37] = '19716939937510582097494459230781640 62';
    PI3: String[37] = '86208998628034825342117067982148086 51';
  var
    Prod: Array[1..120] of Integer;
    A:    Array[1..4]   of Integer;
    PI:   String[111];
    C, CC, I, J, K, L, N, Pr, R, Radius, Code: Integer;

begin
  Write ('Enter N: ');  Readln (N);
  Write ('Enter radius: ');  Readln (Radius);

  { -- Assign digits of PI to Array PI[ ] }
  PI := PI1 + PI2 + PI3;
  L := Length(PI);
  for I := 1 to L do
    Val(Copy(PI, L - I + 1, 1), Prod[I], Code);

  for I := 1 to 3 do A[I] := Radius;
  A[4] := 4;
  C := 0;
  { -- Multiply PI by Radius (3 times) then by 4. }
  for I := 1 to 4 do begin
    for J := 1 to L do begin
      Prod[J] := Prod[J] * A[I] + C;
      C := Prod[J] div 10;
      Prod[J] := Prod[J] - C * 10;
    end;
    while C > 0 do begin
      CC := C div 10;
      Inc(L);
      Prod[L] := C - CC * 10;
      C := CC;
    end;
  end;
  { -- Divide the product by 3. }
  R := 0;
  for I := L downto 1 do begin
    Pr      := Prod[I] + R * 10;
    Prod[I] := Pr div 3;
    R       := Pr - Prod[I] * 3;
  end;
  if Prod[L] = 0 then Dec(L);
  { Display the Volume with the decimal point. }
  for I := L downto 111 - N do begin
    if I = 110 then Write ('.');
    Write (Prod[I]);
  end;
end.
```

```
{3.9}
program Thr9T94;
{ -- This program will display the barcode of an address. }
  const
    Val: Array[1..5] of Byte = (7, 4, 2, 1, 0);
  var
    I, J, L, P, NumBars, CheckDig, Sum, Dig: Byte;
    Addr1, Addr2: String[30];
    BarCode:      String[14];
    Zip4, DPoint: String[4];

begin
  Write ('Enter address 1: ');  Readln (Addr1);
  Write ('Enter address 2: '); Readln (Addr2);

  { -- Extract Zip+4 or Zip from 2nd line of address }
  L := Length (Addr2);
  I := L;
  while Copy(Addr2, I, 1) <> ' ' do I := I - 1;
  if L - I = 10 then
    BarCode := Copy(Addr2, I + 1, 5) + Copy (Addr2, L - 3, 4)
  else
    BarCode := Copy(Addr2, L - 4, 5);

  { -- Extract possible Zip+4 and/or next 2 Delivery points }
  Zip4 := '';
  if Copy(Addr1, 1, 8) = 'P.O. BOX' then
    begin
      L := Length (Addr1);
      I := L;
      while Copy(Addr1, I, 1) <> ' ' do I := I - 1;
      for J := 1 to 4 - (L - I) do
        Zip4 := Zip4 + '0';
      Zip4 := Zip4 + Copy(Addr1, I + 1, L - I);
      DPoint := Copy(Zip4, 3, 2);
    end
  else
    begin
      Zip4 := '0000';
      Addr1 := '0' + Addr1;
      P := Pos (' ', Addr1);
      DPoint := Copy (Addr1, P - 2, 2);
    end;

  if Length(BarCode) = 5 then
    BarCode := BarCode + Zip4;
  BarCode := BarCode + DPoint;

  { -- Calculate Check Digit for 12-digit Barcode and display }
  Sum := 0;
  for I := 1 to 11 do
    Sum := Sum + Ord(BarCode[I]) - 48;
  CheckDig := 10 - (Sum mod 10);
  if CheckDig = 10 then CheckDig := 0;
  Barcode := BarCode + Chr(CheckDig + 48);
```

```
  Writeln (' ': 12, 'DELIVERY POINT BAR CODE = ', BarCode);
  Writeln;

  { -- Display Frame bars and encoded BarCode }
  Write ('!');
  for I := 1 to 12 do begin
    Dig := Ord(BarCode[I]) - 48;
    NumBars := 0;
    if Dig = 0 then  { -- exception for 0 = 7 + 4}
      Dig := 11;
    for J := 1 to 5 do
      if (Dig >= Val[J]) and (NumBars < 2) then
        begin
          Write ('!');
          Dig := Dig - Val[J];
          NumBars := NumBars + 1;
        end
      else
        Write (' ');
  end;  { -- for I }
  Writeln ('!');

  for I := 1 to 62 do Write ('!');
end.
```

```
{3.10}
program Thr10T94;
{ -- This program produces a 3 x 3 magic square. }
  type
    String9  = Array [1..9] of Integer;
  var
    I, Number, FNum, Inc, MNum, Sum:  Integer;
    Num1, Num2, Row, Col, Pos1, Pos2: Integer;
    S: String9;

procedure Permute ({Using}     N:     Integer;
                   {Giving} var S:     String9);
{ -- This procedure will interchange the elements in Array S. }
  var
    I, J, K, Temp: Integer;
    MagicNum:      Boolean;

begin
  If N > 1 then
    begin
      Permute (N - 1, S);
      for I := N-1 downto 1 do begin
        {Interchange the elements in S[N] and S[I] }
        Temp := S[N];  S[N] := S[I];  S[I] := Temp;
        Permute (N - 1, S);
        Temp := S[N];  S[N] := S[I];  S[I] := Temp;
      end;  { -- for I }
    end  { -- if then }
  else
    if (S[Pos1] = Num1) and (S[Pos2] = Num2) then begin
      MagicNum := True;
      { -- Check if Row elements sum to Magic Number. }
      for J := 0 to 2 do
        if S[J*3 + 1] + S[J*3 + 2] + S[J*3 + 3] <> MNum then
          MagicNum := False;
      { -- Check if Column elements sum to Magic Number. }
      if MagicNum then
        for J := 1 to 3 do
          if S[J] + S[J + 3] + S[J + 6] <> MNum then
            MagicNum := False;
      { -- Check if Diagonal elements sum to Magic Number. }
      if MagicNum then
        if  (S[1] + S[5] + S[9] = MNum)
        and (S[3] + S[5] + S[7] = MNum) then begin
          { -- Display the Magic Square. }
          for J := 0 to 2 do begin
            for K := 1 to 3 do
              Write (S[J * 3 + K] :3);
            Writeln;
          end;
          Writeln;
        end;
    end;  { -- if S[Pos1] }
end;  { -- procedure}
```

```
{ -- Main program }
begin
  Write ('Enter first number: ');  Readln(FNum);
  Write ('Enter increment: ');     Readln(Inc);

  Write ('Enter number: ');    Readln (Num1);
  Write ('Enter row, col: ');  Readln (Row, Col);
  Pos1 := (Row - 1) * 3 + Col;
  Write ('Enter number: ');    Readln (Num2);
  Write ('Enter row, col: ');  Readln (Row, Col);
  Pos2 := (Row - 1) * 3 + Col;

  Number := 9;  Sum := 0;
  for I := 1 to Number do begin
    S[I] := FNum + (I - 1) * Inc;
    Sum  := Sum + S[I];
  end;
  MNum := Sum div 3;
  Permute (Number, S);
  Writeln ('MAGIC NUMBER = ', MNum);
end.


{ -- **********   Alternate solution for 3.10  ********** }
program Thr10T94;
{ -- This program produces a 3 x 3 magic square. }
  type
    String9  = Array [1..3, 1..3] of Integer;
  var
    I, J, FNum, Inc, MNum, Sum:         Integer;
    Num1, Num2, Row1, Col1, Row2, Col2: Integer;
    S: String9;

procedure FillRow;
begin
{ -- Determine missing row element from the other two. }
  for I := 1 to 3 do begin
    if (S[I, 1] = 0) and (S[I, 2] > 0) and (S[I, 3] > 0) then
      S[I, 1] := MNum - S[I, 2] - S[I, 3];
    if (S[I, 1] > 0) and (S[I, 2] = 0) and (S[I, 3] > 0) then
      S[I, 2] := MNum - S[I, 1] - S[I, 3];
    if (S[I, 1] > 0) and (S[I, 2] > 0) and (S[I, 3] = 0) then
      S[I, 3] := MNum - S[I, 1] - S[I, 2];
  end;
end;

procedure FillCol;
{ -- Determine missing column element from the other two. }
begin
  for J := 1 to 3 do begin
   if (S[1, J] = 0) and (S[2, J] > 0) and (S[3, J] > 0) then
     S[1, J] := MNum - S[2, J] - S[3, J];
   if (S[1, J] > 0) and (S[2, J] = 0) and (S[3, J] > 0) then
     S[2, J] := MNum - S[1, J] - S[3, J];
```

```
   if (S[1, J] > 0) and (S[2, J] > 0) and (S[3, J] = 0) then
     S[3, J] := MNum - S[1, J] - S[2, J];
  end;
end;


begin
  Write ('Enter first number: ');  Readln(FNum);
  Write ('Enter increment: ');     Readln(Inc);

  Write ('Enter number: ');    Readln (Num1);
  Write ('Enter row, col: ');  Readln (Row1, Col1);
  Write ('Enter number: ');    Readln (Num2);
  Write ('Enter row, col: ');  Readln (Row2, Col2);

  Sum := 0;
  for I := 1 to 3 do
    for J := 1 to 3 do begin
      S[I, J] := 0;
      Sum  := Sum + FNum + ((I-1) * 3 + (J-1)) * Inc;
  end;
  MNum := Sum div 3;  { -- Magic Number }

  S[Row1, Col1] := Num1;
  S[Row2, Col2] := Num2;
  S[2, 2] := Sum div 9;   { -- Middle number is always Sum / 9 }
  { -- Compute the element on the opposite ends of the 2 Nums. }
  S[4-Row1, 4-Col1] := MNum - S[2, 2] - S[Row1, Col1];
  S[4-Row2, 4-Col2] := MNum - S[2, 2] - S[Row2, Col2];

  FillRow;
  FillCol;
  FillRow;

  { -- Display the magic square and magic number. }
  for I := 1 to 3 do begin
    for J := 1 to 3 do
      Write (S[I, J] : 3);
    Writeln;
  end;
  Writeln;
  Writeln ('MAGIC NUMBER = ', MNum);
end.
```